# SCI 4010 COMPUTATIONAL METHODS

MSc Course on Scientific Computing
Hong Kong Baptist University

# Contents

# Chapter 4

# Numerical Solution of Algebraic Systems

In previous chapters we have seen that a wide variety of problems lead to the need of solving linear or nonlinear systems of algebraic equations. For solving a linear system $Ax = b$, a key consideration is the structure of the matrix $A$. In some problems, for example least squares problems, matrix $A$ may be a full matrix; that is, it has few zero elements. On the other hand, we saw that for spline interpolation and certain two-point boundary value problems, $A$ was tridiagonal. And for boundary-value problems of PDEs, $A$ was very large and very sparse (many zero elements). In general, the zero in the matrix $A$ must be utilized as much as possible in order to have efficient algorithms.

There are two general approaches to the solution of linear systems. Direct Methods obtain the exact solution (in real arithmetic) in finitely many operations whereas Iterative Methods generate a sequence of approximations that only converge in the limit to the solution.

Generally, direct methods are best for full or banded matrices whereas iterative methods are best for very large and sparse matrices, especially those arising from PDEs.

## 4.1 Gaussian Elimination

### 4.1.1 Algorithm

We consider the linear system

$$Ax = b$$

where $\boldsymbol{A}$ is a given $n \times n$ matrix assumed to be nonsingular, $\boldsymbol{b}$ is a given column vector and $\boldsymbol{x}$ is the solution vector to be determined.

To describe the algorithm of Gaussian elimination. we write out the linear system in the form

$$\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= a_{1,n+1} \quad (b_1)\cdots(E_1) \\
a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= a_{2,n+1} \quad (b_2)\cdots(E_2)
\end{aligned}$$

$$\vdots$$

$$a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = a_{n,n+1} \quad (b_n)\cdots(E_n)$$

where $b_i$ are represented by $a_{i,n+1}$ for symbolic convenience and $(E_i)$ represents the $i^{\text{th}}$ equation of the system.
Let

$$\tilde{A}^{(0)} = \begin{bmatrix} a_{11}^{(0)} & a_{12}^{(0)} & \cdots & a_{1n}^{(0)} & a_{1,n+1}^{(0)} \\ a_{21}^{(0)} & a_{22}^{(0)} & \cdots & a_{2n}^{(0)} & a_{2,n+1}^{(0)} \\ ]\vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1}^{(0)} & a_{n2}^{(0)} & \cdots & a_{nn}^{(0)} & a_{n,n+1}^{(0)} \end{bmatrix}.$$

**Stage 1**: Provided $a_{11}^{(0)} \neq 0$, then perform

$$(E_i) - \frac{a_{i1}^{(0)}}{a_{11}^{(0)}}(E_1) \to (E_i), \qquad i = 2, 3, \ldots, n$$

At last

$$\tilde{A}^{(0)} \Rightarrow \tilde{A}^{(1)} = [a_{ij}^{(1)}] = \begin{bmatrix} a_{11}^{(0)} & a_{12}^{(0)} & \cdots & a_{1n}^{(0)} & a_{1,n+1}^{(0)} \\ 0 & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} & a_{2,n+1}^{(1)} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & a_{n2}^{(1)} & \cdots & a_{nn}^{(1)} & a_{n,n+1}^{(1)} \end{bmatrix}$$

where

$$a_{ij}^{(1)} = \begin{cases} a_{ij}^{(0)}, & i = 1; \quad j = 1, \ldots, n+1 \\ 0 & j = 1; \quad i = 2, \ldots, n \\ a_{ij}^{(0)} - \left(\dfrac{a_{i1}^{(0)}}{a_{11}^{(0)}}\right) a_{1j}^{(0)}, & i = 2, \ldots, n; \quad j = 2, \ldots, n+1 \end{cases}$$

**Stage k**: At stage $(k-1)$, we obtained

$$\tilde{A}^{(k-1)} = \begin{array}{c} \begin{bmatrix} a_{11}^{(0)} & a_{12}^{(0)} & \cdots & a_{1,k-1}^{(0)} & a_{1,k}^{(0)} & \cdots & a_{1,n+1}^{(0)} \\ 0 & a_{22}^{(1)} & \cdots & a_{2,k-1}^{(1)} & a_{2,k}^{(1)} & \cdots & a_{2,n+1}^{(1)} \\ \vdots & & \ddots & \vdots & \vdots & \ddots & \vdots \\ \vdots & & & a_{k-1,k-1}^{(k-2)} & a_{k-1,k}^{(k-2)} & \cdots & a_{k-1,n+1}^{(k-2)} \\ \vdots & & & 0 & a_{k,k}^{(k-1)} & \cdots & a_{k,n+1}^{(k-1)} \\ \vdots & & & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 & a_{n,k}^{(k-1)} & \cdots & a_{n,n+1}^{(k-1)} \end{bmatrix} \end{array} \begin{array}{c} \\ \\ \\ \\ (E_k) \\ \vdots \\ (E_n) \end{array}$$

Provided $a_{k,k}^{(k-1)} \neq 0$, perform

$$E_i - \frac{a_{i,k}^{(k-1)}}{a_{k,k}^{(k-1)}}(E_k) \Rightarrow E_i, \qquad i = k+1, \ldots, n$$

then we obtain $\tilde{A}^{(k)}$ from $\tilde{A}^{(k-1)}$ where

$$a_{ij}^{(k)} = \begin{cases} 0 & j = k; \quad i = k+1, \ldots, n \\ a_{ij}^{(k-1)} - \dfrac{a_{i,k}^{(k-1)}}{a_{k,k}^{(k-1)}} a_{k,j}^{(k-1)}, & i = k+1, \ldots, n; \quad j = k+1, \ldots, n+1 \\ a_{ij}^{(k-1)}, & \text{others} \end{cases}$$

After the Stage $(n-1)$, we obtain

$$\tilde{A}^{(n-1)} = \begin{bmatrix} a_{11}^{(0)} & a_{12}^{(0)} & \cdots & a_{1n}^{(0)} & a_{1,n+1}^{(0)} \\ 0 & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} & a_{2,n+1}^{(1)} \\ \vdots & & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & a_{n,n}^{(n-1)} & a_{n,n+1}^{(n-1)} \end{bmatrix}$$

This completes the Forward Reduction (or Triangular Reduction) phase of the Gaussian Elimination.

**Stage n: Backward Substitution**

$$x_n = \frac{a_{n,n+1}^{(n-1)}}{a_{n,n}^{(n-1)}}$$

$$x_i = \frac{a_{i,n+1}^{(i-1)} - \displaystyle\sum_{j=i+1}^{n} a_{ij}^{(i-1)} x_j}{a_{ii}^{(i-1)}}, \qquad i = n-1, \ldots, 1$$

**Remark**

If $a_{k,k}^{(k-1)} = 0$, then to search for the first nonzero entry on the $k^{th}$ column of $\tilde{A}^{(k-1)}$ from $a_{k+1,k}^{(k-1)}$ to $a_{n,k}^{(k-1)}$ (because of nonsingularity of $A$, there must be at least one nonzero entry). If $a_{p,k}^{(k-1)} \neq 0$, for some $p \geq k+1$, then

$$(E_k) \leftrightarrow (E_p)$$

and continue the procedure.

**Algorithm**

**Input** number of equations $n$; augmented matrix $A = [a_{ij}]$ where $1 \leq i \leq n, \quad 1 \leq j \leq n+1$

**Output** solution $x_1, \ldots, x_n$ or message of no unique solution.

**Step 1** For $k = 1, \ldots, n-1$ do Step 2-4 (forward reduction)

**Step 2** Let $p$ be the smallest integer with $k \le p \le n$ and $a_{pk} \ne 0$.
If no $p$ can be found, then Output the message of
"no unique solution" and Stop.

**Step 3** If $p \ne k$ then perform $(E_p) \leftrightarrow (E_k)$

**Step 4** For $i = k+1, \dots, n$ do Steps 5 and 6

**Step 5** Set $l_{ik} = \dfrac{a_{ik}}{a_{kk}}$

**Step 6** For $j = k+1, \dots, n+1$ do $a_{ij} = a_{ij} - l_{ik} \cdot a_{kj}$

**Step 7** If $a_{nn} = 0$, Output "no unique solution"; Stop.

**Step 8** Set $x_n = \dfrac{a_{n,n+1}}{a_{nn}}$ (Start backward substitution)

**Step 9** For $i = n-1, \dots, 1$ do $x_i = \dfrac{(a_{i,n+1} - \sum_{j=i+1}^{n} a_{ij} x_j)}{a_{ii}}$

**Step 10** Output $(x_1, x_2, \dots, x_n)$; Stop.

### 4.1.2 $LU$ Factorization

In Gaussian elimination, the Stage 1 is equivelence to multiplying the matrix $\tilde{A}^{(0)}$ by the matrix

$$L_1 = \begin{bmatrix} 1 & & & \\ -l_{21} & 1 & & \\ \vdots & & \ddots & \\ -l_{n1} & & & 1 \end{bmatrix}$$

where

$$l_{i1} = \frac{a_{i1}^{(0)}}{a_{11}^{(0)}}, \qquad i = 2, \dots, n$$

That is

$$\tilde{A}^{(1)} = L_1 \tilde{A}^{(0)}$$

Similarly, the Stage $k$ is equivalent to

$$\tilde{A}^{(k)} = L_k \tilde{A}^{(k-1)}$$

where

$$L_k = \begin{bmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & -l_{k+1,k} & 1 & & \\ & & \vdots & & \ddots & \\ & & -l_{n,k} & & & 1 \end{bmatrix}$$

with

$$l_{i,k} = \frac{a_{i,k}^{(k-1)}}{a_{k,k}^{(k-1)}}, \qquad i = k+1, \dots, n\,.$$

Hence, computing in this fashion, we have

$$L_{n-1}\cdots L_2 L_1 \tilde{A}^{(0)} = \tilde{A}^{(n)} = \begin{bmatrix} a_{11}^{(0)} & a_{12}^{(0)} & \cdots & a_{1n}^{(0)} & a_{1\,n+1}^{(0)} \\ & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} & a_{2\,n+1}^{(1)} \\ & & \ddots & \vdots & \vdots \\ & & & a_{nn}^{(n-1)} & a_{n\,n+1}^{(n-1)} \end{bmatrix}.$$

Let

$$U = \begin{bmatrix} a_{11}^{(0)} & a_{12}^{(0)} & \cdots & a_{1n}^{(0)} \\ & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} \\ & & \ddots & \vdots \\ & & & a_{nn}^{(n-1)} \end{bmatrix};$$

$$\hat{L} = L_{n-1}L_{n-2}\cdots L_1\,;$$
$$L = \hat{L}^{-1} = L_1^{-1}L_2^{-1}\cdots L_{n-1}^{-1}\,.$$

It follows from $\hat{L}A = U$ that

$$A = \hat{L}^{-1}U = LU\,.$$

It can be verified that

$$L_k^{-1} = \begin{bmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & l_{k+1,k} & 1 & & \\ & & \vdots & & \ddots & \\ & & l_{n,k} & & & 1 \end{bmatrix}$$

and

$$L = L_1^{-1}\cdots L_{n-1}^{-1} = \begin{bmatrix} 1 & & & & \\ l_{21} & 1 & & & \\ l_{31} & l_{32} & 1 & & \\ \vdots & \vdots & \ddots & \ddots & \\ l_{n1} & l_{n2} & \cdots & l_{n,n-1} & 1 \end{bmatrix}.$$

The Gaussian elimination for solving $Ax = b$ is mathematically equivalent to the three-step process:

1. Factor $\boldsymbol{A = LU}$
2. Solve $\boldsymbol{Ly = b}$
3. Solve $\boldsymbol{Ux = y}$

The above algorithm of forward reduction has found

$$U = L_{n-1}\cdots L_2 L_1 A$$

and

$$y = L_{n-1} \cdots L_2 L_1 b = [a_{1,n+1}^{(0)} \;\; a_{2,n+1}^{(1)}, \cdots, a_{n,n+1}^{(n-1)}]^T$$

In the forward reduction process, the matrix $\boldsymbol{L}$ is also obtained by putting $l_{ij}$ together. Solving $\boldsymbol{Ux} = \boldsymbol{y}$ is just the process of backward substitution.

Note that in this algorithm the $a_{ij}^{(k)}$ are overwritten onto the same storage spaces occupied by the original elements $a_{ij}$. The multiplier $l_{ij}$ can be written into the corresponding storage space for $a_{ij}$ which is no longer needed after $l_{ij}$ is computed.

### 4.1.3 Direct Factorization

We can regard the Direct Factorization $A = LU$ as a system of $n^2$ equations with $n^2$ unknowns $l_{i,j}$ and $u_{i,j}$. It can be solved directly in the following manner. From the multiplication of

$$\begin{bmatrix} 1 & & & & \\ l_{21} & 1 & & & \\ l_{31} & l_{32} & 1 & & \\ \vdots & \vdots & \ddots & \ddots & \\ l_{n1} & l_{n2} & \cdots & l_{n,n-1} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & \cdots & \cdots & u_{1n} \\ & u_{22} & \cdots & \cdots & u_{2n} \\ & & \ddots & & \vdots \\ & & & \ddots & \vdots \\ & & & & u_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & & \vdots \\ \vdots & \vdots & & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & \cdots & a_{nn} \end{bmatrix}$$

we obtain $n^2$ equation

$$\begin{array}{llll} u_{11} = a_{11}, & u_{12} = a_{12} & \cdots\cdots\cdots & u_{1n} = a_{1n} \\ l_{21}u_{11} = a_{21}, & l_{21}u_{12} + u_{22} = a_{22} & \cdots\cdots\cdots & l_{21}u_{1n} + u_{2n} = a_{2n} \\ l_{k1}u_{11} = a_{k1}, & l_{k1}u_{12} + l_{k2}u_{22} = a_{k2} & \cdots & \left\{ \begin{array}{ll} \displaystyle\sum_{m=1}^{j} l_{km}u_{mj} = a_{kj}, & j \leq k-1 \\ \displaystyle\sum_{m=1}^{k-1} l_{km}u_{mj} + u_{kj} = a_{kj}, & j \geq k \end{array} \right. \end{array}$$

**Algorithm for getting $L$ and $U$**

**Step 1** For $j = 1, \ldots, n$ set $u_{1j} = a_{1j}$

**Step 2** For $i = 2, \ldots, n$ set $l_{i1} = \dfrac{a_{i1}}{u_{11}}$

**Step 3** For $k = 2, \ldots, n-1$ do Step 4-5

**Step 4** For $j = k, \ldots, n$ set $u_{k,j} = a_{k,j} - \displaystyle\sum_{m=1}^{k-1} l_{k,m}u_{mj}$

**Step 5** For $i = k+1, \ldots, n$ set $l_{i,k} = \dfrac{(a_{ik} - \sum_{m=1}^{k-1} l_{im}u_{mk})}{u_{kk}}$

**Step 6** $\quad u_{nn} = a_{n,n} - \sum_{m=1}^{n-1} l_{n,m} u_{m,n}$ .

The order of computation

$$
\begin{array}{lll}
 & \text{first row of } U & \rightarrow \text{ first column of } L \\
\rightarrow & \text{second row of } U & \rightarrow \text{ second column of } L \\
\rightarrow & \cdots & \\
\rightarrow & (n-1)^{th} \text{ row of } U & \rightarrow \ (n-1)^{th} \text{ column of } L \\
\rightarrow & u_{nn} &
\end{array}
$$

**Remark**:

The algorithm of direct factorization takes advantage of the condensed calculation of the inner product by the row vector of $\boldsymbol{L}$ and column vector of $\boldsymbol{U}$. But in some case, if $u_{kk} = 0$ the computation may break down.

It can be shown that, if $\boldsymbol{A}$ is diagonally dominent or positive definite, then the algorithm never break down.

**Operation Counts**

The major part of the work is in the forward reduction by the arithmetic statement

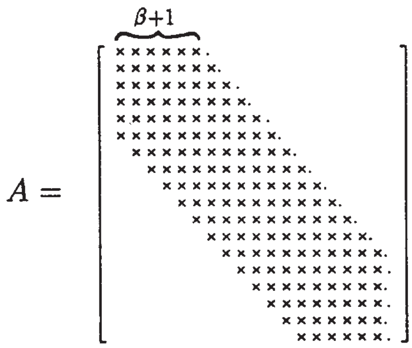
$$a_{ij} = a_{ij} - i_{ik} a_{kj}$$

In the computing stage $k$, there are $(n-k)^2$ multiplications and additions. Hence, the total number of multiplications and addition is

$$\sum_{k=1}^{n-1} (n-k)^2 \approx \frac{n^3}{3}$$

### 4.1.4 Banded Matrices

For many matrices that arise in practice, particularly in the solution of differential equations, the elements of the matrix are primarily zero. The simplest examples of this are the tridiagonal matrices which are special cases of banded matrices.

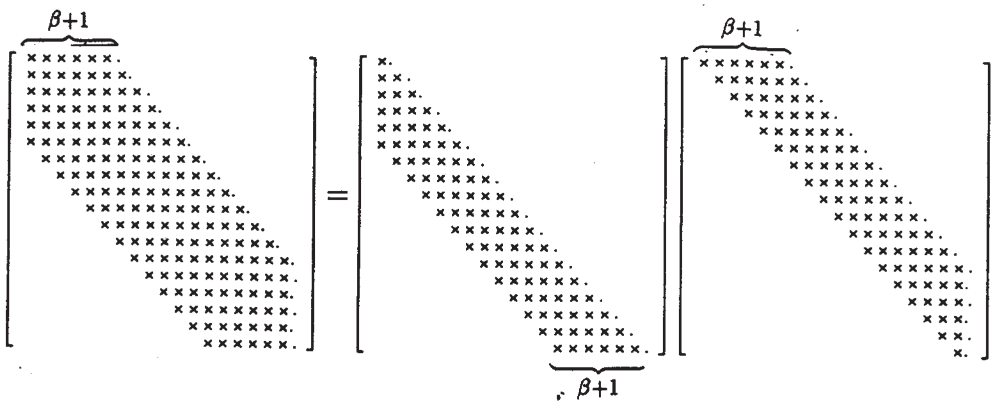


**Definition 4.1** *A matrix is called* banded *matrix if its elements satisfy for all $i, j$,*

$$a_{ij} = 0 \qquad \text{when } |j - i| \geq \beta + 1$$

*where $\beta$ is called the* semi-bandwidth.

The Gaussian elimination algorithm for banded matrices benefits by not having to deal with the zero elements outside of the band. For a banded matrix, the factors $\boldsymbol{L}$ and $\boldsymbol{U}$ of the $\boldsymbol{LU}$ decomposition retain the same bandwidth, i.e.,




- The number of additions (or multiplications) required is approximately $n\beta^2$.
- The total storage required, including the right-hand side and the solution vector is $n(2\beta+3)$.

### Determinants and Inverses

By the $\boldsymbol{LU}$ decomposition of $\boldsymbol{A}$, and using the fact that the determinant of a product of two matrices, we have

$$\begin{aligned} \det A &= \det(LU) = \det L \times \det U \\ &= u_{11}u_{22}\cdots u_{nn} \end{aligned}$$

since $L$ has 1's on its main diagonal and so $\det L = 1$.
Let $\boldsymbol{e_i}$ be the vector with 1 in the $i^{th}$ position and zeros elsewhere. Then $\boldsymbol{e_i}$ is the $i^{th}$ column of the identity matrix $\boldsymbol{I}$.

Let

$$A^{-1} = [x_1, x_2, \ldots, x_n]\,.$$

Then it follows from $AA^{-1} = I$ that

$$A[x_1, x_2, \ldots, x_n] = [e_1, e_2, \ldots, e_n]$$

or equivalently

$$Ax_i = e_i, \qquad i = 1, \ldots, n\,.$$

For solving these $n$ systems with different right hand side vectors, this $\boldsymbol{LU}$ decomposition of $\boldsymbol{A}$ is needed only once.

We stress that one does not wish to solve a system $\boldsymbol{Ax} = \boldsymbol{b}$ by computing $\boldsymbol{A}^{-1}$ and then forming $\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}$. This would generally require much more work than just solving the system.

### 4.1.5 Fill

In this first stage of Gaussian elimination, the element $a_{ij}$ is modified to become $a_{ij}^{(1)}$ by

$$a_{ij}^{(1)} = a_{ij} - \frac{a_{i1}}{a_{11}} a_{1j}$$

If $a_{ij} = 0$, but $a_{i1}$ and $a_{ij}$ are both non-zero, then $a_{ij}^{(1)} \neq 0$ and a non-zero element has been introduced into a position of the matrix that was originally zero. That is called fill. It can occur in any stage of Gaussian elimination.

As a simple example of the effect of fill, consider the matrix

$$A = \begin{bmatrix} \times & \times & & & \times \\ \times & \times & \times & & \\ & \ddots & \ddots & \ddots & \\ & & \times & \times & \times \\ \times & & & \times & \times \end{bmatrix}$$

which arose for a two-point boundary problem with periodic boundary conditions.

In each stage of elimination, the last row and last column also have one fill. In terms of the $\boldsymbol{LU}$ decomposition, we have

$$L = \begin{bmatrix} \times & & & & \\ \times & \times & & & \\ & \ddots & \ddots & & \\ & & \ddots & \ddots & \\ \times & \times & \cdots & \times & \times \end{bmatrix} \qquad U = \begin{bmatrix} \times & \times & & & \times \\ & \times & \times & & \times \\ & & \ddots & \ddots & \vdots \\ & & & \times & \times \\ & & & & \times \end{bmatrix}.$$

#### Sherman-Morrision Formula

To solve a system with a coefficient matrix of the above form not only increases the operation count cosiderably over a tridiagonal matrix, but also requires a much more complicated code. We next consider, by use of a so called Sherman-Morrision Formula, that decreases the operation count slightly but, more importantly, utilizes a tridiagonal matrix code.

Consider the linear system

$$\begin{bmatrix} b_{11} & b_{12} & & & c \\ b_{21} & b_{22} & b_{23} & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & b_{n-1,n} \\ d & & & b_{n,n-1} & b_{n,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_n \end{bmatrix} = b$$

Let $\boldsymbol{B}$ be the tridiagonal part of the coefficient matrix $\boldsymbol{A}$, then, $\boldsymbol{A}$ may be written as

$$A = B + ce_1e_n^T + de_ne_1^T \,,$$

where $\boldsymbol{e_i}$ is the vector with 1 in the $i^{th}$ position and zero elsewhere. Let

$$\begin{aligned} u &= ce_1 + de_n \,, \\ v &= e_1 + e_n \,. \end{aligned}$$

Therefore we have

$$A = T + uv^T \,,$$

where

$$T = B + \mathrm{diag}(-c, 0, \ldots, 0, -d) \,.$$

It is observed that $\boldsymbol{T}$ is still a tridiagonal matrix.

Let $\boldsymbol{C}$ be any nonsingular matrix. It is easy to check that

$$\begin{aligned} (C + uv^T)^{-1} &= C^{-1} - \alpha^{-1}C^{-1}uv^TC^{-1}, \\ \alpha &= 1 + v^TC^{-1}u \end{aligned}$$

The equality is called Sherman-Morrison formula. Hence, the solution of

$$(C + uv^T)x = b$$

can be written as

$$\begin{aligned} x &= (C + uv^T)^{-1}b \\ &= C^{-1}b + \alpha^{-1}C^{-1}uv^TC^{-1}b \\ &= y + \alpha^{-1}z(v^Ty), \qquad \alpha = 1 + v^Tz \end{aligned}$$

where $y = C^{-1}b$ and $z = C^{-1}u$ or equivalently to say, $y$ and $z$ are solution of

$$\begin{aligned} Cy &= b \,, \\ Cz &= u \,, \end{aligned}$$

respectively.

Thus, for our linear system where $C = T$ is a tridiagonal system, the solution of $Ax = b$ would consist of the steps:

$$\begin{array}{ll} \text{Solve:} & Ty = b \\ & Tz = u \\ \text{Form:} & \alpha = 1 + v^T z \\ & x = y - \alpha^{-1}(v^T y) z \end{array}$$

It requires only a code for solving tridiagonal system.

**Fill for the Poisson Matrix**

The matrix obtained by discretizing Poisson's equation in a unit square domain can be expressed as follows:

$$A = \begin{bmatrix} T & -I & & & \\ -I & T & -I & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & -I \\ & & & -I & T \end{bmatrix}.$$

$\boldsymbol{A}$ is a banded matrix with bandwidth $n$ (as $h = \dfrac{1}{n+1}$) although it only has five non-zero diagonals.

When the Gaussian elimination is used, the fill in $\boldsymbol{LU}$ factorization is almost as much as if the matrix $\boldsymbol{A}$ were a full banded matrix; the sparse structure of $\boldsymbol{A}$ within the band has essentially been lost because of the fill.

Corrspondingly, the amonunt of work to carry out the elimination also is almost as much as if the matrix were a full banded matrix. Since the order if $\boldsymbol{A}$ is $n^2$ and the bandwidth is $n$, hence the operation count for elimination (or $\boldsymbol{LU}$ factorization) is approximately $n^4$.

### 4.1.6 Pivoting Strategies

In Gaussian elimination, the elements $a_{kk}^{(k-1)}$ are called pivot elements. It is often necessary to perform row interchanges even when the pivot elements are not zero, to reduce the round-off error associated with finite-digit arithmetic.

If $a_{kk}^{(k-1)}$ is small in magnitude compared to $a_{ik}^{(k-1)}$ $i$ $(i > k)$ the multiplier

$$l_{ik} = \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}}$$

will have magnitude much larger than 1. Then a rounding error in the terms $a_{k,j}^{(k-1)}$ may be much enlarged into $a_{ij}^{(k)}$ $(i > k)$ when computing

$$a_{ij}^{(k)} = a_{ij}^{(k-1)} - l_{ik} a_{kj}^{(k-1)}$$

Also, when performing the backward substitution for $x_k$,

$$x_k = \frac{(a_{k,n+1}^{(k-1)} - \sum_{j=k+1}^{n} a_{kj}^{(k-1)} x_j)}{a_{kk}^{(k-1)}}$$

any rounding error in the numerator will be dramatically increased when dividing by $a_{kk}^{(k-1)}$.

**Example**: Consider the linear system

$$\begin{aligned} E_1: &\quad 0.0030000x_1 + 5.914x_2 &= 59.17 \\ E_2: &\quad 5.291x_1 - 6.130x_2 &= 46.78 \end{aligned}$$

which has the exact solution $x_1 = 10.00$ and $x_2 = 1.000$

Assume that Gaussian elimination will be performed using four-digit arithemetic with rounding. The first pivot element is a small number $a_{11}^{(0)} = 0.003000$, and its associated multiplier

$$l_{21} = \frac{5.291}{0.003000} = 1764$$

Performing $(E_2 - l_{21} \times E_1) \to E_2$, we have

$$-1.043 \times 10^5 x_2 = -1.044 \times 10^5$$

Backward substitution yields

$$x_2 = 1.001$$

and

$$x_1 = \frac{59.17 - (59.14) \times (1.001)}{0.003000} = -10.00$$

We see that because of the small pivot $a_{11}^{(0)}$, the small error of 0.001 on $x_2$ multiplied by

$$\frac{59.14}{0.0030000}$$

the approximate of $x_1$ is ruined far from the exact solution.

**Reconsider the algorithm**

First step: $(E_2) \leftrightarrow (E_1)$, then

$$l_{21} = \frac{0.003000}{5.291} = 0.0005670$$

$(E_2 - l_{21} \times E_1) \to (E_2)$: $59.14\, x_2 = 59.14\,.$

Backward substitution: $x_2 = 1.000$

$$x_1 = \frac{46.78 + 6.130 \times 1.000}{5.291} = 10.00$$

The approximate has no error! The technique used is called Partial Pivoting.

**Partial Pivoting**

At the $k^{th}$ stage of the elimination process an interchange of rows is made to place in the main diagonal position the element of largest absolute value from the $k^{th}$ column on or below the main diagonal.

**Scaled-Column Pivoting**

The linear system

$$\begin{aligned} E_1 : \quad & 30.00x_1 + 5.914 \times 10^5 x_2 &= 5.917 \times 10^5 \\ E_2 : \quad & 5.291x_1 - 6.130x_2 &= 46.78 \end{aligned}$$

is the same as that in the last example except that the first equation have been multiplied by $10^4$. But now by Pivoting Strategy described above, the first pivot should be 30.00. Then the multiplier

$$l_{21} = \frac{5.291}{30.00} = 0.1764$$

and leads to the system

$$\begin{aligned} E_1 : \quad & 30.00x_1 + 5.914 \times 10^5 x_2 &= 5.917 \times 10^5 \\ E_2 : \quad & -1.043 \times 10^5 x_2 &= -1.044 \times 10^5 \end{aligned}$$

which has the same inaccurate solution as in the first case of last example: $x_2 = 1.001, \quad x_1 = -10.00$.

A technique known as scaled-column pivoting is appropriate to overcome the difficulty. The first step in this procedure is to define for each row a scale factor $s_i$ by

$$s_i = \max_{1 \le j \le n} |a_{ij}|$$

Then the first pivot is determined by choosing the $a_{i1}$ with

$$\frac{|a_{i1}|}{s_i} = \max_{1 \le k \le n} \frac{|a_{k1}|}{s_k}$$

and performing $(E_1) \leftrightarrow (E_i)$.

The effect of scaling is to ensure that the largest element in each row has a magnitude 1 before the comparison for pivoting. The scaling is done only for comparison purpose, so the division by scaling factors produces no round-off error in the system.

**Diagonally Dominant and Positive Definite Matrices**

Although for general nonsingular matrices it is necessary to use the partial pivoting strategy, there are some types of matrices for which it is known that no interchanges are necessary. The most important of these are diagonally dominant matrices and positive definite matrices. In both cases, it is safe to use Gaussian elimination with no interchanges at all.

Not needing to interchange is especially advantages for banded matrices (in particular, tridiagonal), and it is a fortunate fact that many banded matrices arising in applications, especially from differential equations, are either diagonally dominant or positive definite.

### 4.1.7 Numerically Unstable and Ill Conditioning

There are two possibilities as to how rounding error can effect the computing solution. The first is an accumulation of rounding errors during a large number of arithmetic operations. For example, if $n = 1000$, the operation count of a full matrix will be on the order of $n^3 = 10^9$. Even though the error in each individual operation may be small, the total buildup could be large. In fact, this potential accumulation of rounding error is not as serious as one might expect.

The second possibility involves catastroplic rounding errors, such as the case of the first algorithm used in the example of last sction. If an algorithm has this unfortunate characteristic, it is called numerically unstable and is not suitable as a general method.

**Verification of Numerical Stability**

Let $\tilde{\boldsymbol{x}}$ be the computed solution of an algorithm for the linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$. Substitute $\tilde{\boldsymbol{x}}$ to the equations, we obtain the residual

$$r = b - A\tilde{x}$$

If $\dfrac{||r||_2}{||b||_2}$ is small, for example

$$\frac{||r||_2}{||b||_2} < n\beta^{-t}$$

where $t$ is the digits in mantissa, $\beta$ is the base of number system and $n$ is the order of matrix $\boldsymbol{A}$, then we say the algorithm is numerically stable, otherwise, say numerically unstable.

The first algorithm in the example is numerically unstable, because the computed solution $x_1 = 1.001 \quad x_2 = -10.00$ leads

$$\frac{||r||_2}{||b||_2} = \frac{12.27}{75.43} = 0.16 \gg 2 \times 10^{-4}$$

The second algorithm with partial pivoting is numerically stable, because the computed solution $x_1 = 1.000$ and $x_2 = 10.00$ leads $||r||_2 = 0$.

**Ill-Conditioning**

Even though the Gaussian elimination algorithm with partial pivoting has proved to be an efficient and reliable method in practive, it may fail to compute accurate solutions of systems of equations that are, ill-conditioned. A linear system of equations is said to be "ill-conditioned" if small changes in the elements or right-hand side $\boldsymbol{b}$ cause large changes in the solution. In this case no numerical method can be expected to produce an accurate solution. The reason is: a stable algorithm can only guarantee the resudual small, i.e.,

$$A\tilde{x} = b - r$$

where $\tilde{\boldsymbol{x}}$ and $\boldsymbol{r}$ are computed solution and residual respectively. But now, if the system is ill conditioned, a small changes $\boldsymbol{r}$ in $\boldsymbol{b}$ may cause large changes in the solution $\boldsymbol{x}$ of $\boldsymbol{Ax} = \boldsymbol{b}$.

**Condition Number**

We now consider a way of measuring the ill-conditioning of a matrix $\boldsymbol{A}$ (as well as the system $\boldsymbol{Ax} = \boldsymbol{b}$) be mears of norms. Suppose first that $\boldsymbol{x}$ is the solution of $\boldsymbol{Ax} = \boldsymbol{b}$ and that $\tilde{\boldsymbol{x}} = \boldsymbol{x} + \boldsymbol{\Delta x}$ is the solution of the system with residual $\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{A\tilde{x}}$, i.e.,

$$A(x + \Delta x) = b - r$$

It also means a changes $\Delta b = -r$ cause a change $\Delta x$ in the solution $x$. Since $\boldsymbol{Ax} = \boldsymbol{b}$, it follows that $A\Delta x = \Delta b (= -r)$ or

$$\Delta x = A^{-1}\Delta b$$

Thus

$$||\Delta x|| \leq ||A^{-1}||\,||\Delta b||$$

which shows that the change in the solution due to a change in the right-hand side (or the residual) is bounded by $||A^{-1}||$. Thus, a small change in $\boldsymbol{b}$ may cause a large change in $x$ if $||A^{-1}||$ is large. The notion of "large" is always relative, however, and it is more useful to deal with relative change $\frac{||\Delta x||}{||x||}$. From $Ax = b$, it follows that

$$||b|| \leq ||A||\,||x||$$

Combining this with $||\Delta x|| \leq ||A^{-1}||\,||\Delta b||$ yields

$$||b||\,||\Delta x|| \leq ||A||\,||A^{-1}||\,||\Delta b||\,||x||$$

or, equivalently

$$\frac{||\Delta x||}{||x||} \leq ||A||\,||A^{-1}||\frac{||\Delta b||}{||b||}$$

This inequality shows that the relative change in $x$ is bounded by the relative change in $b$, multiplied by $||A||\,||A^{-1}||$. Hence, the quantity $||A||\,||A^{-1}||$ is of great importance and is called the condition number of A (with repsect to the norm $||\cdot||$ being used) and denoted by $\text{cond}(A)$.

Note first that

$$||A||\,||A^{-1}|| \geq ||A \cdot A^{-1}|| = ||I|| = 1$$

it means that always $\text{cond}(A) \geq 1$. If $\text{cond}(A)$ is close to 1, then small relative changes in the data can lead to only small relative changes in the solution. In this case, we say that the problem is well-conditioned.

Let $\tilde{x} = x + \Delta x$ be an approximate solution of a stable algorithm, then $\dfrac{||r||}{||b||}$ is small and the inequality

$$\frac{||\Delta x||}{||x||} \le \mathrm{cond}(A) \cdot \frac{||r||}{||b||}$$

holds. Thus, if the problem is also well-conditioned, the relative error $\dfrac{||\Delta x||}{||x||}$ is also small.

## 4.2 Iterative Methods

We have seen that when we apply Gaussian elimination to a large sparse system of equations, such as those arising from discretizing Poisson's equation, fill occurs and increases the operation count and the storage requirements. For these systems, iterative methods are often the only possible methods of solution due to their less requirement of storage and sometimes also faster than elimination. They usually have better vectorization and parallelization properties. The largest area for the application of iterative methods is to the linear systems arising in the numerical solution of PDEs. For this type of problems, the coefficient matrix is of high order, usually over $10^4$, and sparse which means only a small percentage of the coefficients are nonzero.

### 4.2.1 Simultaneous Displacements (Gauss-Jacobi)

Consider the linear system $Ax = b$ and assume that

$$a_{ii} \neq 0, \qquad i = 1, \dots, n$$

Taking an initial guess $x^0$, and then compute

$$x_i^{(1)} = \frac{1}{a_{ii}}(b_i - \sum_{j \neq i} a_{ij} x_j^{(0)}), \qquad i = 1, \dots, n\,.$$

We can write this in matrix-vector notation. Let $D = \mathrm{diag}(a_{11}, \dots, a_{nn})$ and $B = D - A$. Then

$$x^{(1)} = D^{-1}(b + Bx^{(0)})$$

and the entire sequence is defined by

$$x^{(k+1)} = D^{-1}(b + Bx^{(k)}), \qquad k = 0, 1, \dots$$

This iterative procedure is called Gauss-Jacobi method.

### 4.2.2 Successive Displacements (Gauss-Seidel)

A closely related iteration is derived from the following observation. After $x_1^{(1)}$ is computed in

$$x_1^{(1)} = \frac{1}{a_{11}}(b_1 - a_{12}x_2^{(0)} - a_{13}x_3^{(0)} - \cdots - a_{1n}x_n^{(0)})$$

it is available to use in the computation of $x_2^{(1)}$, and it is natural to use this new value rather than $x_1^{(0)}$. Hence, we may compute

$$x_2^{(1)} = \frac{1}{a_{22}}(b_1 - a_{11}x_1^{(1)} - a_{13}x_3^{(0)} - a_{14}x_4^{(0)} - \cdots - a_{1n}x_n^{(0)})$$

and

$$x_i^{(1)} = \frac{1}{a_{ii}}(b_i - \sum_{j<i} a_{ij}x_j^{(1)} - \sum_{j>i} a_{ij}x_j^{(0)}), \qquad i = 1, \ldots, n$$

which is the first step in the Gauss-Seidel iteration.

To write this iteration in matrix-vector form, let $-L$ and $-U$ denote the strictly lower and upper triangular parts of $A$; that is, both $L$ and $U$ have zero main diagonals and

$$A = D - L - U$$

Then the system $Ax = b$ is equivalent to

$$Dx = Lx + Ux + b$$

or

$$x = D^{-1}(b + Lx + Ux)$$

The entire sequence of Gauss-Seidel iterates is defined by

$$x^{(k+1)} = D^{-1}(b + Lx^{(k+1)} + Ux^{(k)})$$

or equivalent to

$$x^{(k+1)} = (D - L)^{-1}(Ux^{(k)} + b), \qquad k = 0, 1, \ldots$$

### 4.2.3 Storage Requirements

In iterative methods, storage is needed in general only for the non-zero elements of $A$, the right hand side $b$, and the current and previous iterates $x^{(k)}$ and $x^{(k+1)}$. In the Gauss-Seidel method, one vector suffices for both $x^{(k)}$ and $x^{(k+1)}$ since new components of $x^{(k+1)}$ may be written into the corresponding positions of $x^{(k)}$ as soon as they are computed.

As opposed to a direct method such as Gaussian elimination, there is no need to allow additional storage for fill, since the original matrix is not modified.

#### Application to Laplace's Equation

Consider the application of these iterative methods to Laplace's equation on a square. The finite difference equations are given in the form (assumed $f_{ij} = 0$)

$$4u_{ij} - u_{i+1,j} - u_{i-1,j} - u_{i\,j+1} - u_{i\,j-1} = 0, \qquad i, j = 1, \ldots, N$$

where $u_{0,j}$, $u_{N,j}$, $u_{i,0}$, $u_{i,N}$ are known from the boundary conditions.
Then the Gauss-Jacobi method gives the sequence of iterates as

$$u_{ij}^{(k+1)} = \frac{1}{4}(u_{i+1,j}^{(k)} + u_{i-1,j}^{(k)} + u_{i,j+1}^{(k)} + u_{i,j-1}^{(k)}) \qquad k = 0, 1, \ldots$$

For the Gauss-Seidel method, different ordering of the equations corresponds to a different iterative process. If we order the grid points left to right and bottom to top, then a typical Gauss-Seidel step is

$$u_{ij}^{(k+1)} = \frac{1}{4}(u_{i-1,j}^{(k+1)} + u_{i,j-1}^{(k+1)} + u_{i+1,j}^{(k)} + u_{i,j+1}^{(k)})$$

This new approximation at the $(i, j)$ grid point is again an average of the approximations at the four surrounding grid points, as Gauss-Jacobi method does, but now using two old values and two new values.
Because the above two methods give a new iterate by averaging current values at neighboring points, they were known historically as relaxation methods.

### 4.2.4 Convergence

In general, an iterative method for solving a linear system $Ax = b$ can be defined as

$$x^{(k+1)} = Hx^{(k)} + d, \qquad k = 0, 1, \ldots$$

where

$$x = Hx + d$$

is an equivalent system to $Ax = b$.
Different methods lead to different $H$. For examples:

$$\begin{aligned} \text{Gauss-Jacobi} \ : \ & H = D^{-1}B = D^{-1}(L + U), \\ & d \ = D^{-1}b \\ \text{Gauss-Seidel} \ : \ & H = (D - L)^{-1}U \\ & d \ = (D - L)^{-1}b \end{aligned}$$

Let

$$e^{(k)} = x^{(k)} - x$$

be the error of $x^{(k)}$ to the exact solution $x$, then $e^{(k)}$ satisfies

$$e^{(k+1)} = He^{(k)}$$

Using the formula recurrently, we have

$$e^{(k)} = H^k e^{(0)}$$

In terms of an arbitrary norm, we have the following inequality

$$\begin{aligned} ||e^{(k)}|| & \leq ||H^k|| \, ||e^{(0)}|| \\ & \leq ||H||^k \, ||e^{(0)}|| \end{aligned}$$

Hence, if

$$||H|| < 1$$

then

$$\lim_{k\to\infty} ||e^{(k)}|| = 0$$

i.e.,

$$\lim_{k\to\infty} x^{(k)} = x$$

Thus, $||H|| < 1$ is a sufficient condition for convergence of an iterative method.

A sufficient and necessary condition is given in terms of eigenvalues of $H$. Assume that $H$ has $n$ linearly independent eigenvectors $v_1, v_2, \ldots, v_n$ with corresponding eigenvalues $\lambda_1, \ldots, \lambda_n$. The initial error $e^{(0)}$ can then be expressed as some linear combination of the eigenvectors:

$$e^{(0)} = c_1 v_1 + c_2 v_2 + \cdots c_n v_n$$

Thus,

$$\begin{aligned} e^{(k)} &= H^k e^{(0)} = \sum_{i=1}^{n} c_i H^k v_i \\ &= \sum_{i=1}^{n} c_i \lambda_i^k v_i \end{aligned}$$

It can be seen that if and only if

$$|\lambda_i| < 1, \qquad i = 1, \ldots, n$$

then

$$\lim_{k\to\infty} e^{(k)} = 0$$

Define the spectral radius

$$\rho(H) = \max_{1\le i\le n} |\lambda_i|$$

Then the sufficient and necessary condition for convergence is

$$\rho(H) < 1$$

Assume that

$$|\lambda_i| < |\lambda_n|, \qquad i = 1, \ldots, n-1$$

Then when $k$ sufficient large,

$$e^{(k)} \approx c_n \lambda_n^k v_n$$

It means each additional iteration reduces the error by approximately the factor $|\lambda_n|$. Thus, $\rho(H)$ is called the asymptotic convergence factor.

We stat this basic convegence theorem as:

**Theorem 4.1** *For any initial vector $x^{(0)}$, if and only if $\rho(H) < 1$, the iterates converge to the solution $x$. Moreover, the smaller $\rho(H)$, the faster is the asymptotic rate of convergence.*

From the general convergence theorem, we can obtain some useful results for certain class of matrices.

**Theorem 4.2** *Assume that the matrix $A$ is strictly diagonally dominant:*

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}|, \qquad i = 1, \ldots, n\,.$$

*Then both the Gauss-Jacobi and Gauss-Seidel iterations are convergent for any initial vector $x^{(0)}$.*

**Theorem 4.3** *Assume that the matrix $A$ is symmetric and positive definite, then the Gauss-Seidel method is convergent.*

### 4.2.5 SOR Method

Although Gauss-Jacobi and Gauss-Seidel methods are convergent, the rate of convergence is very slow. For example, for Poisson equation on a unit square with $h = \frac{1}{45}$, the error in each iteration of the Gauss-Seidel method will decrease asymptotically only by a factor of about 0.995, i.e., $\rho(H) = 0.995$. As $h \to 0$, $\rho(H) = 1 - \alpha h^2$.

In certain cases, it is possible to accelerate considerably the Gauss-Seidel method by the so called successive overrelaxation (SOR) method.

**Algorithm of SOR**: Given the current approximation $x^{(k)}$, we first compute the Gauss-Seidel interate

$$\hat{x}_i^{(k+1)} = \frac{1}{a_{ii}}(b_i - \sum_{j<i} a_{ij} x_j^{(k+1)} - \sum_{j>i} a_{ij} x_j^{(k)})$$

as an intermediate value, and then take the final value of the new approximation to the $i^{th}$ component to be

$$x_i^{(k+1)} = x_i^{(k)} + \omega(\hat{x}_i^{(k+1)} - x_i^{(k)})$$

Here $w$ is a parameter that has been introduced to accelerate the convergence.

Combining the above two formulas, we can get

$$x_i^{(k+1)} = (1-\omega)x_i^{(k)} + \frac{\omega}{a_{ii}}(b_i - \sum_{j<i} a_{ij} x_j^{(k+1)} - \sum_{j>i} a_{ij} x_j^{(k)})$$

and then rearrange the formula into the form

$$a_{ii} x_i^{(k+1)} + \omega \sum_{j<i} a_{ij} x_j^{(k+1)} = (1-\omega) a_{ii} x_i^{(k)} - \omega \sum_{j>i} a_{ij} x_j^{(k)} + \omega b_i$$

Write it in matrix-vector form as

$$Dx^{(k+1)} - \omega L x^{(k+1)} = (1-\omega) D x^{(k)} + \omega U x^{(k)} + \omega b$$

If $D$ has non-zero diagonal elements, the matrix $D - \omega L$ is nonsingular. We may write

$$x^{(k+1)} = (D - \omega L)^{-1}[(1-\omega)D + \omega U]x^{(k)} + \omega(D - \omega L)^{-1}b$$

Let $H = (D - \omega L)^{-1}[(1-\omega)D + \omega U]$ and $d = \omega(D - \omega b)^{-1}b$, we have,

$$x^{(k+1)} = H_\omega x^{(k)} + d\,.$$

According to the general convergence theorem mentioned in last section, $\rho(H_\omega) < 1$ is the necessary and sufficient condition for convergence. We would like to choose the parameter $\omega$ so as to optimize the convergence rate, i.e., to minimize the value of $\rho(H_\omega)$. In general this is a very difficult problem. Here we only mention some results related to Poisson equation.

Let matrix $H_1 = (D - L)^{-1}U$ denote the Gauss-Seidel iteration matrix of the discrete Poisson equation. It can be shown that the optimum value of $\omega$, denoted by $\omega_0$, is given in terms of $\rho(H_1)$, by

$$\omega_0 = \frac{2}{1 + \sqrt{1 - \rho(H_1)}}$$

The correspoinding value of the spectral radius of $H_\omega$ is

$$\rho(H_{\omega_0}) = \omega_0 - 1\,.$$

For the Poisson equation on a square with grid size $h$

$$\rho(H_1) = \cos^2 \pi h$$

then

$$\omega_0 = \frac{2}{1 + \sqrt{1 - \cos^2 \pi h}}$$

and

$$\rho(H_{\omega_0}) = \omega_0 - 1 = \frac{1 - \sqrt{1 - \cos^2 \pi h}}{1 + \sqrt{1 - \cos^2 \pi h}}\,.$$

When $h = \dfrac{1}{45}$, $\rho(H_1) \approx 0.995$, and

$$\rho(H_{\omega_0}) \approx 0.87$$

As $h \to 0$, $\rho(H_1) = 1 - \pi^2 h^2$ and

$$\rho(H_{\omega_0}) = 1 - \pi h\,.$$

For general domain rather than rectangle, it is difficult to know $\rho(H_1)$ in advance. There are some "adaptive methods" that help to approximate a good value of $\omega$ as the SOR iteration proceeds. A simple one is as following: At the beginning, the SOR iteration proceeds with $\omega = 1$, (that is Gauss-Seidel method). After some steps, we may have

$$\frac{||x^{(k+1)}||_2}{||x^k||_2} \approx \rho(H_1)\,.$$

But, in order to obtain a good approximation to $\rho(H_1)$, the number $k$ should be sufficiently large.

**Operation Counts**

Let $e^{(k)} = x - x^{(k)}$, where $x$ is the exact solution of the system $Ax = b$ and $x^{(k)}$ is the $k^{th}$ iterate of SOR method. Because $x^{(k)}$ satisfies

$$x^{(k)} = H_\omega x^{(k-1)} + d$$

and $x$ satisfies

$$x = H_\omega x + d$$

which is the equivalent system of $Ax = b$, thus $e^{(k)}$ satisfies

$$\begin{aligned} e^{(k)} &= H_\omega e^{(k-1)} \\ &= H_\omega^k e^{(0)} \end{aligned}$$

In term of Euclidian norm, we have

$$||e^{(k)}||_2 \le ||H_\omega^k||_2 \, ||e^{(0)}||_2$$

or

$$\frac{||e^{(k)}||_2}{||e^{(0)}||_2} \le ||H_\omega^k||_2$$

It can be shown that as $k$ sufficiently large, $[\rho(A)]^k \approx ||A^k||_2$ for arbitrary matrix $A$. Hence,

$$\frac{||e^{(k)}||_2}{||e^{(0)}||_2} \le [\rho(H_\omega)]^k$$

for sufficiently large $k$.

If we require the error $e^{(k)}$ to attain an accuracy of

$$\frac{||e^{(k)}||_2}{||e^{(0)}||_2} \le 10^{-\lambda}$$

the number of iteration $k$ should satisfy

$$\rho(H_\omega)]^k = 10^{-\lambda}$$

or equivalently

$$k = \frac{-\lambda \ln_e 10}{\ln_e \rho(H_\omega)} .$$

For discrete Poisson equation, if an optimum parameter $\omega_0$ is chosen, then

$$\rho(H_{\omega_0}) = 1 - \alpha h .$$

When $h$ is sufficiently small

$$\ln_e \rho(H_{\omega_0}) \approx -\alpha h .$$

Substitute to the above formula of $k$, we obtain

$$k = (\frac{\lambda \ln_e 10}{\alpha}) \cdot \frac{1}{h}.$$

It means that, the iteration number $k$ is proportional inversely to the grid size $h$. On the other hand, it is obvious that the operation counts of each iteration is proportional to the number of grid points, that is $O(h^{-2})$ if in terms of $h$. Hence, the total operation counts is of order $O(h^{-3})$.

For the typical problem of Poisson equation on a unit square, the grid size $h = \frac{1}{n}$ where $n$ is the number of one row. Thus, in terms of $n$, the operation counts of SOR method is of order $O(n^3)$ if the optimum parameter $\omega_0$ can be chosen. Compared with Gauss elimination method, which has the operation counts of order $O(n^4)$, the SOR method is lower cost.

### 4.2.6 Conjugate Gradient Method

#### Minimization Methods

A number of iterative methods for solving linear systems of equations can be derived as minimization methods. If $A$ is symmetric and positive definite, then the quadratic function

$$\begin{aligned} q(x) &= \frac{1}{2}\sum_{i,j=1}^{n} a_{ij}x_ix_j - \sum_{j=1}^{n} b_jx_j + c \\ &= \frac{1}{2}(Ax, x) - (b, x) + c \\ &= \frac{1}{2}x^TAx - x^Tb + c \end{aligned}$$

has a unique minimizer which is the solution of

$$Ax = b$$

This can be showed by differential calculus and also can be showed in the following manner: Let $x^* = A^{-1}b$, then

$$q(x) = \frac{1}{2}(A(x - x^*), (x - x^*)) + c - \frac{(Ax^*, x^*)}{2}$$

The equality can be checked easily. Because $A$ is positive definite, therefore $q(x)$ has a unique minimizer and the minimizer is $x^* = A^{-1}b$. Thus, to find the solution of

$$Ax = b$$

where $A$ is positive definite, is equivalent to find the minimizer of the quadratic function $q(x)$.

Many minimization methods can be written in the form

$$x_{k+1} = x_k - \alpha_k p_k, \qquad k = 0, 1, \ldots$$

Given the direction vector $p_k$, one way to choose the scalar $\alpha_k$ is to minimize $q(x)$ along $x_k - \alpha p_k$ so that

$$q(x_k - \alpha_k p_k) = \min_{\alpha} q(x_k - \alpha p_k)$$

For fixed $x_k$ and $q_k$, $q(x_k - \alpha p_k)$ is a quiadratic function of $\alpha$. By differentiation it leads to

$$\alpha_k = -\frac{(p_k, r_k)}{(Ap_k, p_k)} \tag{4.1}$$

where $r_k = b - Ax_k$ .

Although there are many other ways to choose the $\alpha_k$, we will use only the above one and concentrate on different choices of the direction vectors $p_k$. One simple choice is $p_k = r_k = b - Ax_k$, which gives the method of

**Steepest descent**:

$$x_{k+1} = x_k - \alpha_k (b - Ax_k)\,.$$

This is also known as Richardson's method and is closely related to Gauss-Jacobi method. It's convergence is usually very slow. Another simple strategy is to take

$$p_k = e_{k+1}, \qquad k = 0, \ldots, n-1$$

and $\alpha_k$ are chosen by (4.1). Then, every $n$ steps of the iteration is equivalent to one Gauss-Seidel iteration on the system $Ax = b$.

**Conjugate Gradient Method**

Assume that $A$ is symmetric positive definite. Two vectors $p_i$ and $p_j$ are called $A$-orthogonal or Conjugate with respect to $A$, if

$$(Ap_i, p_j) = 0$$

The importance of the concept "conjugate" is presented in the following theorem.

**Theorem 4.4** *If $A$ is a symmetric positive definite matrix of order $n \times n$, and $p_0, \ldots, p_{n-1}$ are conjugate each other with respect to $A$, then for any initial vector $x_0$, the iterates*

$$x_{k+1} = x_k - \alpha_k p_k, \qquad k = 0, 1, \ldots$$

*where $\alpha_k$ is chosen by (4.1), converge to the solution of $Ax = b$ in no more than $n$ steps.*

**Algorithm**

Choose $x_0$, set $p_0 = r_0 = b - Ax_0$
For $k = 0, 1, \dots$

$$\alpha_k = -\frac{(r_k, r_k)}{(Ap_k, p_k)} \qquad \left(= -\frac{(p_k, r_k)}{(Ap_k, p_k)}\right)$$

$x_{k+1} = x_k - \alpha_k p_k$
$r_{k+1} = r_k + \alpha_k A p_k$
if $||r_{k+1}||_2^2 \geq \epsilon$, continue

$$\beta_k = \frac{(r_{k+1}, r_{k+1})}{(r_k, r_k)} \qquad \left(= -\frac{(p_k, Ar_{k+1})}{(p_k, Ap_k)}\right)$$

$p_{k+1} = r_{k+1} + \beta_k p_k$

In the above alogrithm, we have given two equivalent forms of $\alpha_k$ and $\beta_k$. The second form of $\alpha_k$ is (4.1) and the second form of $\beta_k$ is the condition that $p_{k+1}$ be conjugate to $p_k$. The first forms given are most suitable for computation since only the inner products $(r_k, r_k)$ and $(Ap_k, p_k)$ are required. The step of $r_{k+1}$ computes the residual recursively rather than directly since

$$\begin{aligned} r_{k+1} &= b - Ax_{k+1} = b - A(x_k - \alpha_k p_k) \\ &= r_k + \alpha_k A p_k \end{aligned}$$

Because $Ap_k$ is already known, this saves the multiplication $Ax_{k+1}$. It can be proved that the $p_k$ $(k = 0, 1, \dots, n-1)$ obtained by the algorithm are conjugate with respect to the matrix $A$. Hence, if no rounding error, we can obtain the solution of $Ax = b$ in no more than $n$ steps. In fact, the rounding error must exist. Thus it is impossible $r_n = b - Ax_n = 0$. If after one cycle of iteration, i.e., $n$ steps of iteration, still $||r_n||_2^2 \geq \epsilon$, then we may take $x_n$ as a new initial vector $x_0$ and begin the next cycle until $||r_k||_2^2 \leq \epsilon$.

**Convergence**

Let $\kappa$ be the condition number of $A$ in 2-norm, i.e.,

$$\kappa = ||A||_2 \, ||A^{-1}||_2$$

and

$$\gamma = \frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}.$$

It can be shown that

$$||x^* - x_k||_2 \leq 2\sqrt{\kappa}\gamma^k ||x^* - x_0||_2.$$

Since $\gamma < 1$, $x_k$ converges to $x^*$.

For typical problem of discrete Poisson equation on a unit square, the $\kappa = O(h^{-2})$ and then

$$\gamma = 1 - \alpha h, \qquad \alpha > 0i,$$

the same order of convergence rate as SOR method with optimum parameter $\omega_0$. Thus the operation counts is also $O(n^3)$ where $n = h^{-1}$.

**Comment** The high efficiency of SOR method is very restrictive, mainly for Poisson equation with choosing an optimum parameter $\omega_0$. The C-G method is highly efficient in common use and without the trobule of choosing parameter.

## 4.3 Nonlinear Equations and Newton's Method

We now turn to the solution of nonlinear equation and begin with root-finding for a single nonlinear equation.

### 4.3.1 Root-finding for a single equation

Finding one or more roots of an equation

$$f(x) = 0$$

where $f(x)$ is at least assumed continuous but may be no explicit expression, is one of the commonly occuring problems in practice.

#### Bisection Method

Assume that $f(x)$ is continuous on $[a, b]$ and satisfies

$$f(a) \cdot f(b) < 0$$

Using the intermediate value theorem, the function $f(x) = 0$ has at least one root in $[a, b]$

**Algorithm**: Assumed that $f(x)$ is continuous on $[a, b]$ where $f(a)$ and $f(b)$ have opposite signs.

**Input** : endpoints $a, b$; tolerance TOL
**Output** : approximate solution $x$

- **Step 1** : Set $x = a + \dfrac{b-a}{2}$
- **Step 2** : If $f(x) = 0$ or $\dfrac{b-a}{2} < TOL$ then Output $(x)$; Stop
- **Step 3** : If $f(a) \cdot f(x) > 0$ then Set $a = x$, else Set $b = x$
- **Step 4** : Return Step 1

#### Comments on Bisection Method

**Advantage** : very weak assumption, always converge
**Drawback** : after $n$ times of iteration

$$|x_n - x^*| \leq \frac{b-a}{2^n}, \qquad x^* \text{ — exact solution,}$$

the convergent speed is very slow compared with other methods.

**Newton's Method**

The Newton's method is one of the most powerful and well-known numerical methods for solving non-linear equations. Suppose that the function $f$ is twice continuously differential on $[a,b]$, and the initial value $x_0$ is close to the desired root $x^*$ of $f(x)=0$ . Consider the first degree Taylor polynomial for $f(x)$ expanded about $x_0$

$$f(x) = f(x_0) + (x-x_0)f'(x_0) + \frac{(x-x_0)^2}{2}f''(\xi)$$

where $\xi$ lies between $x$ and $x_0$
Take $x=x^*$. If $|x^*-x_0|$ is small, then the higher term $(x^*-x_0)^2$ could be negligible and

$$f(x_0) + (x^*-x_0)f'(x_0) \approx f(x^*) = 0$$

Solving for $x^*$ and obtain

$$x^* \approx x_0 - \frac{f(x_0)}{f'(x_0)}$$

It means that

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

should be a better approximation to $x^*$ than is $x_0$.

Continue to iterate in a same manner, i.e.,

$$x_n = x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})}, \qquad n = 1, 2, \ldots$$

we get a sequence $\{x_n\}$, which should be one better than one for approximating $x^*$.

**Graphical Interpretation**

The equation of tangent of

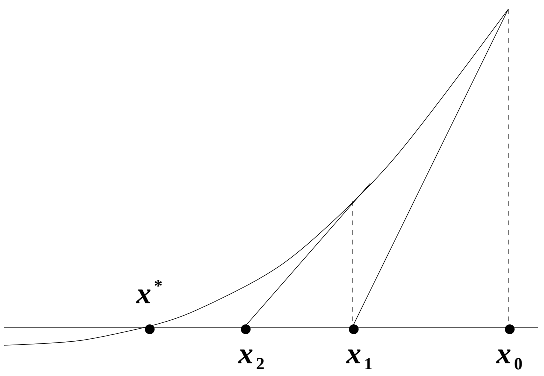
$$y = f(x)$$

at the point $(x_0, f(x_0))$ is

$$\frac{y - f(x_0)}{x - x_0} = f'(x_0)$$

or

$$y = f(x_0) + f'(x_0)(x - x_0)$$




So that

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

is the intersect of the tangent and the $x$-axis. It can be seen that $x_1$ is closer to $x^*$ than $x_0$. The Newton's method is often derived from this geometrical point of view and also called tangent method.

**Convergence Theorem**

Assume that $f(x)$, $f'(x)$ and $f''(x)$ are continuous for $x$ in some neighborhood of a root $x^*$ and $f(x^*) \neq 0$. Let $I = [x^* - \delta, x^* + \delta]$

$$M = \frac{1}{2} \frac{\max_{x \in I} |f''(x)|}{\min_{x \in I} |f'(x)|}$$

Then if anititial value $x_0$ is close to $x^*$ within

$$|x_0 - x^*| < \frac{1}{M}$$

the iteraters $x_n$ will converge to $x^*$. Moreover

$$\lim_{n \to \infty} \frac{x^* - x_{n+1}}{(x^* - x_n)^2} = -\frac{f''(x^*)}{2f'(x^*)} .$$

**Remarks**:

- The theorem not only affirms the convergence for a "good" initial estimate, but also give the convergent speed of

  $$|x^* - x_{n+1}| \approx C|x^* - x_n|^2$$

  where

  $$C \approx \left| \frac{f''(x^*)}{f'(x^*)} \right|$$

- Whenever the iterates $x_n$ of an iterative method satisfy

  $$|x^* - x_{n+1}| \leq C|x^* - x_n|^\alpha$$

  we say that the method has an **order $\alpha$ of convergent speed**. Newton's method is of second order. The bisection method is of linear order (or first order) of which the iterates $x_n$ satisfy

  $$|x^* - x_{n+1}| \leq \frac{1}{2}|x^* - x_n| .$$

**Secant Method**

Sometimes the $f(x)$ has no explicit expression such as the case we have mentioned in shooting method. Then we cannot calculate the derivative of $f(x)$. But, we still can use the following revision of Newton's method. It is known that for small increment $\Delta x$

$$f'(x) \approx \frac{f(x + \Delta x) - f(x)}{\Delta x} .$$

In Newton's method

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \qquad n = 0, 1, \dots,$$

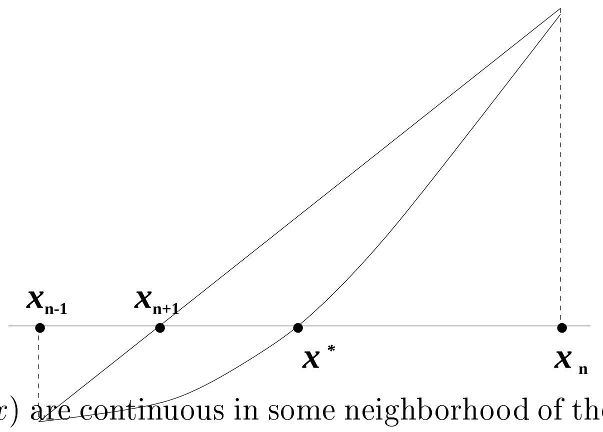
if $\dfrac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}$ is used instead of $f'(x_n)$, we get a formula

$$x_{n+1} = x_n - f(x_n) \cdot \frac{(x_n - x_{n-1})}{f(x_n) - f(x_{n-1})}, \qquad n = 1, \dots .$$

That is so-called secant method, because $x_{n+1}$ is the root of the secant line determined by $(x_{n-1},\, f(x_{n-1}))$ and $(x_n,\, f(x_n))$:

$$\frac{y - f(x_n)}{x - x_n} = \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}.$$

For the secant method, two initial values $x_0$, $x_1$ are needed to start with.




**Convergence**

Assume that $f(x)$, $f'(x)$ and $f''(x)$ are continuous in some neighborhood of the desired root $x^*$ and asuume $f'(x^*) \neq 0$. Let $I = [x^* - \epsilon,\, x^* + \epsilon]$ and

$$M = \frac{1}{2} \frac{\max\limits_{x \in I} |f''(x)|}{\min\limits_{x \in I} |f'(x)|}$$

Then if the initial values $x_0$ and $x_1$ are chosen within $I$ and satisfy

$$|x - x_j| < \frac{1}{M}, \qquad j = 0, 1$$

the iterates $x_n$ of the secant method will converge with order

$$\alpha = \frac{1 + \sqrt{5}}{2} \approx 1.618.$$

**Comparison of Newton's and Secant Methods**

- The conditions for convergence are same

- Newton's method requires evaluations of $f(x_n)$ and $f'(x_n)$ per iterate, whereas the secant method requires only one function evaluation. Newton's method is more expensive per iteration. On the other hand, Newton's method converges more rapidly [ order $\alpha = 2$ of Newton's method vs. $\alpha = 1.618$ of secant method ] and consequently it will require fewer iterations to attain a given desired accuracy.
- An analysis shows that if the time to evaluate $f'(x)$ is more than 44% of that necessary to evaluate $f(x)$, then the secant method is more efficient.

### 4.3.2 Newton's Method for Nonlinear System

Most of true mathematical models of real world problems are nonlinear systems involving not only one unknown. A system of nonlinear equations has the following general form

$$\begin{cases} f_1(x_1, x_2, \ldots, x_n) = 0 \\ f_2(x_1, x_2, \ldots, x_n) = 0 \\ \qquad \vdots \\ f_n(x_1, x_2, \ldots, x_n) = 0 \end{cases}$$

Let vector $X$ and function vector $F$ represent

$$X = [x_1, x_2, \ldots, x_n]^T$$

and

$$F(X) = [f_1(X), f_2(X), \ldots, f_n(X)]^T$$

Then $F(X)$ can be seen as a mapping from $R^n$ into $R^n$ and the nonlinear system has the form

$$F(X) = 0 \quad \text{(zero vector)}$$

In root-finding of a single nonlinear equation, we have learnt the Newton's method. It's main idea is to use a first degree Taylor polynomial to approximate the given function $f(x)$. This idea can be generalized to nonlinear system.

**Linearlization of a Nonlinear System**

Consider the nonlinear system

$$F(X) = 0$$

or

$$\begin{cases} f_1(x_1, x_2, \ldots, x_n) = 0 \\ f_2(x_1, x_2, \ldots, x_n) = 0 \\ \qquad \vdots \\ f_n(x_1, x_2, \ldots, x_n) = 0 \end{cases}$$

Suppose $X^0 = (x_1^0, x_2^0, \ldots, x_n^0)^T$ is a good initial guess to the exact solution $X^* = (x_1^*, x_2^*, \ldots, x_n^*)^T$, a.e.,

$$\delta X = X^* - X^0 = (\delta x_1, \delta x_2, \ldots, \delta x_n)^T$$

is small in norm. Then

$$\begin{aligned} 0 &= f_i(x_1^*, x_2^*, \ldots, x_n^*) = f_i(x_1^0 + \delta x_1, x_2^0 + \delta x_2, \ldots, x_n + \delta x_n) \\ &= f_i(x_1^0, x_2^0, \ldots, x_n^0) + \sum_{k=1}^{n} \frac{\partial f_i}{\partial x_k}(x_1^0, x_2^0, \ldots, x_n^0) \cdot \delta x_k + \text{higher terms}, \end{aligned}$$

for $i = 1, 2, \ldots, n$. In matrix-vector form, we have

$$\begin{bmatrix} \frac{\partial f_1}{\partial x_1}(X^0), & \frac{\partial f_1}{\partial x_2}(X^0), & \cdots & \frac{\partial f_1}{\partial x_n}(X^0) \\ \frac{\partial f_2}{\partial x_1}(X^0), & \frac{\partial f_2}{\partial x_2}(X^0), & \cdots & \frac{\partial f_2}{\partial x_n}(X^0) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1}(X^0), & \frac{\partial f_n}{\partial x_2}(X^0), & \cdots & \frac{\partial f_n}{\partial x_n}(X^0) \end{bmatrix} \begin{bmatrix} \delta x_1 \\ \delta x_2 \\ \vdots \\ \delta x_n \end{bmatrix} + \begin{bmatrix} f_1(X^0) \\ f_2(X^0) \\ \vdots \\ f_n(X^0) \end{bmatrix} \approx 0$$

or

$$DF(X^0) \cdot \delta X + F(X^0) \approx 0$$

where $\delta X = [\delta x_1, \delta x_2, \ldots, \delta x_n]^T$ and $DF(X)$ is the so-called Jacobian matrix of $F(X)$. Hence

$$\delta X \approx -[DF(X^0)]^{-1} F(X^0)$$

method is convergent under some smoothness assumptions of $F(X)$. Due to $X^* = X^0 + \delta X$, we have

$$X^* \approx X^0 - [DF(X^0)]^{-1} F(X^0)$$

It means that the vector

$$X^1 = X^0 - [DF(X^0)]^{-1} F(X^0)$$

will be better than $X^0$ as an approximate to the solution $X^*$.

**Algorithm**

Newton's method for a single equation $f(x) = 0$:

$$x_{k+1} = x_k - [f'(x_k)]^{-1} f(x_k), \qquad k = 0, 1, \ldots$$

Newton's method for a nonlinear system $F(X) = 0$

$$X^{k+1} = X^k - [DF(X^k)]^{-1} F(X^k)$$

just use Jacobian matrix in place of derivative!

**Convergence**

If an initial vector $X^0$ is chosen sufficiently close to the exact solution $X^*$, the Newton's method is convergent under some smoothness assumptions of $F(X)$. Furthermore, the iterates $\{X^k\}$ satisfy

$$||X^* - X^{k+1}||_2 \leq \lambda ||X^* - X^k||_2^2$$

for some constant $\lambda > 0$. It means that the Newton's method for nonlinear system also has order 2 of convergent speed.

**Weakness of Newton's Method**

- Each iteration from $X^k$ to $X^{k+1}$, a Jacobian atrix is required. That is, $n^2$ partial derivatives of the $n$ component functions of $F$ be evaluated. This difficulty can be overcome by using finite difference approximations, for example,

  $$\frac{\partial f_i}{\partial x_k}(x_1, \ldots, x_n) \approx \frac{1}{\Delta x_k}[f_i(x_1, \ldots, x_k + \Delta x_k, \ldots x_n) - f_i(x_1, \ldots, x_k, \ldots, x_n)]$$

  where $\Delta x_k$ is small in absolute value. But, still requires that $n^2$ scalar functional evaluations be performed.

- From $X^k$ to $X^{k+1}$, a $n \times n$ linear system with full coefficient matrix has to be solved. It is required in general $O(n^3)$ operations.

### 4.3.3 Broyden's Method

In this section, we consider a generalization of the secant method known as Broyden's method. The method requires only $n$ scalar functional evaluations per iteration and need not to solve a linear system. The number of arithmetic operations is reduced to $O(n^2)$. It belongs to a class of methods called quasi-Newton.

**Main Idea**

Suppose that an initial approximation $X^0$ is given. We calculate the next approximation $X^1$ by Newton's method

$$X^1 = X^0 - [DF(X^0)]^{-1} F(x^0)$$

or, if it is inconvenient to determine $DF(X^0)$ exactly, we use finite difference approximation. The secant method for single nonlinear equation use the approximation

$$f'(x_1) \approx \frac{f(x_1) - f(x_0)}{x_1 - x_0}$$

as a replacement for $f'(x_1)$ in Newton's method. For nonlinear systems, $X^1 - X^0$ is a vector, and the corresponding quotient is undefined.

However, the method proceeds similarly in that we replace the matrix $DF(X^1)$ in Newton's method by a matrix $A_1$ with the property that

$$A_1(X^1 - X^0) - F(X^1) - F(X^0)$$

since

$$DF(X^1)(X^1 - X^0) \approx F(X^1) - F(X^0)$$

The above equation does not define a unique matrix. For $DF(X^1) \approx DF(X^0)$, we require additional conditions

$$A_1 Z = DF(X^0)Z, \qquad \text{whenever}(X^1 - X^0)^T Z = 0$$

It can be checked that

$$A_1 = DF(X^0) + \frac{[F(X^1) - F(X^0) - DF(X^0)(X^1 - X^0)](X^1 - X^0)}{||X^1 - X^0||_2^2}$$

satisfies all above conditions.

It is this matrix that is used in place of $DF(X^1)$ to determine $X^2$

$$X^2 = X^1 - A_1^{-1}F(X^1)$$

The method is then repeated to determine $X^3$, using $A_1$ in place of $A_0 = DF(X^0)$ and with $X^2$ and $X^1$ in place of $X^1$ and $X^0$. In general, once $X^i$ has been determined, $X^{i+1}$ is computed by

$$\begin{cases} A_i & = A_{i-1} + \dfrac{Y^i - A_{i-1}S^i}{||S^i||_2^2}(S^i)^T \\ Y^i & = F(X^i) - F(X^{i-1}), \qquad S^i = X^i - X^{i-1} \\ X^{i+1} & = X^i - A_i^{-1}F(X^i) \end{cases}$$

If $Z^i = A_i^{-1}F(X^i)$ has to be calculated by solving linear system

$$A_i Z^i = F(X^i)$$

the method is not much improved.

### Avoidable of Solving Linear System

Recall the Sherman-Morrision formula: If $A \in R^{n\times n}$, nonsingular, $X$, $Y \in R^N$ and $Y^T A^{-1} X \neq -1$, then

$$(A + XY^T)^{-1} = A^{-1} - \frac{A^{-1}XY^TA^{-1}}{1 + Y^TA^{-1}X}$$

Using this formula, we have the following algorithm to compute $A_i^{-1}$ from $A_{i-1}^{-1}$. By letting

$$\begin{aligned} A & = A_{i-1} \\ X & = \frac{(Y^i - A_{i-1}S^i)}{||S^i||_2^2} \\ Y & = S^i \\ A_i & = (A + XY^T) \end{aligned}$$

we have

$$
\begin{aligned}
A_i^{-1} &= A_{i-1}^{-1} - \frac{A_{i-1}^{-1}\left(\frac{Y^i - A_{i-1}S^i}{||S^i||_2^2}(S^i)^T\right)A_{i-1}^{-1}}{1 + (S^i)^T A_{i-1}^{-1}\left(\frac{Y^i - A_{i-1}S^i}{||S^i||_2^2}\right)} \\
&= A_{i-1}^{-1} - \frac{(A_{i-1}^{-1}Y^i - S^i)(S^i)^T A_{i-1}^{-1}}{(S^i)^T A_{i-1}^{-1} Y^i}
\end{aligned}
$$

This computation involves only matrix-vector multiplication at each step and therefore requires only $O(n^2)$ arithmetic operations.

## Convergence

The disadvantage of the Broyden's method is that the quadratic convergence of Newton's method is lost, being replaced by a convergence called superlinear.

$$
\lim_{i\to\infty} \frac{||X^{i+1} - X^*||_2}{||X^i - X^*||_2} = 0
$$

In most applications, the reduction to superlinear convergence is a more than acceptable trade-off, for the decrease in the amount of computation. An additional disadvantage of Broyden's method is that, unlike Newton's method, they are not self-correcting. To overcome this difficulty, we could take $X^n$, after $n$ steps of iteration, to be a new initial approximation $X^0$ and restart a new iterative cycle.